

# $k$ -Means for Streaming and Distributed Big Sparse Data

Artem Barger

artem@barger.net

Dan Feldman

dannyf.post@gmail.com

University of Haifa

## Abstract

We provide the first streaming algorithm for computing a provable approximation to the  $k$ -means of sparse Big Data. Here, sparse Big Data is a set of  $n$  vectors in  $\mathbb{R}^d$ , where each vector has  $O(1)$  non-zeroes entries, and  $d \geq n$ . E.g., adjacency matrix of a graph, web-links, social network, document-terms, or image-features matrices.

Our streaming algorithm stores at most  $\log n \cdot k^{O(1)}$  input points in memory. If the stream is distributed among  $M$  machines, the running time reduces by a factor of  $M$ , while communicating a total of  $M \cdot k^{O(1)}$  (sparse) input points between the machines.

Our contribution is a deterministic algorithm for computing a *sparse  $(k, \varepsilon)$ -coreset*, which is a weighted subset of  $k^{O(1)}$  input points that approximates the sum of squared distances from the  $n$  input points to every  $k$  centers, up to  $(1 \pm \varepsilon)$  factor, for any given constant  $\varepsilon > 0$ . This is the first such coreset of size independent of both  $d$  and  $n$ .

Existing algorithms use coresets of size at least polynomial in  $d$ , or project the input points on a subspace which diminishes their sparsity, thus require memory and communication  $\Omega(d) = \Omega(n)$  even for  $k = 2$ .

Experimental results on synthetic and real public dataset shows that our algorithm boost the performance of such given heuristics even in the off-line setting. Open access to our implementation is also provided [6].

## 1 Background

**Clustering.** For a given similarity measure, clustering aims at partitioning a given set of points into coherent subset. There is a lot of different clustering techniques, but most prominent and popular is Lloyd's algorithm or the  $k$ -means algorithm [27, 4]. The input to the classical Euclidean  $k$ -means problem is a set of  $n$  points in  $\mathbb{R}^d$ , and the goal is to compute a set of  $k$ -centers (also points in  $\mathbb{R}^d$ ) that minimizes the sum of squared distances over each input point to its nearest center.

More generally, there are some constraints on the location of the centers. For example, in the *discrete  $k$ -mean* the set of centers must be subset of the input points. This version is preferable for example, when each input point is sparse (i.e., have small number of non-zeroes coordinates), and we wish that the centers will also be sparse. In other applications, such as in computer vision, the input is a set of images and we wish to compute representative  $k$  images (centers). Since it is not clear how to interpret the sum of a vector that represents an image of a cat with a vector that represents an image of a dog, it is natural to ask that the set of  $k$  centers will be a subset of the input images. In facility location problems [17, 11], the input points represent the location of clients, and the centers are called facilities. In this case we might have constraints that some of the centers will be closer to some clients.

However, most of the existing clustering algorithms, especially the ones that support streaming or large distributed data sets, assume that the dimension is significantly smaller than the number of input points. Otherwise, techniques such as PCA/SVD [25, 15] or Johnson-Lindenstrauss [21] are used for reducing the dimensionality of the input. However, such projections turn sparse data into dense data, and the  $k$ -means of the projected points are no longer subset of the input or sparse [15]. In fact, few projected (dense) points

in a high dimensional space might take more memory than the complete original dataset, e.g. in Wikipedia document-term  $n \times d$  matrix, where  $d \gg n$ .

**Sparse Big Data.** We consider sparse Big Data as a set of  $n$  vectors in  $\mathbb{R}^d$ , where each vector has  $O(1)$  non-zeroes entries, and both the number of vectors  $n$  and their dimensionality  $d$  is asymptotically large. Hence, we cannot store all the vectors, or general linear combinations of vectors, in memory. Instead, we allow to scan these vectors in a streaming fashion, while using memory that is only logarithmic in  $n$  or  $d$ .

In particular, unlike previous related work, this paper handle the case  $d \geq n$  (e.g.  $d = n^n$ ), where most of those papers assume  $d \ll n$ . Our algorithm also supports input such as in NoSQL or Matlab sparse matrices, where the row  $i$  is given as a stream of pairs  $(j_1, val_1), (j_2, val_2), \dots$  where  $(j_m, val_m)$  means that the  $j_m$ th entry of the row is  $val_m$ , and the rest are zeroes by default.

Sometimes a cloud or a large set of machines are available for handling Big Data. In this case, the data is distributed among them and we wish not only to process the data in parallel, but also to minimize communication between the machines. Finally, GPUs (Graphical Processing Units) are used to boost the performance of the algorithm, but only if the algorithm uses a limited set of simple operations that are supported by the GPU.

All these models are supported by our algorithm, via the framework of sparse coresets that is explained below.

**Coresets** Given a set of  $n$  points in  $\mathbb{R}^d$ , and an error parameter  $\varepsilon > 0$ , a coreset in this paper is a small set of weighted points in  $\mathbb{R}^d$ , such that the sum of squared distances from the original set of points to any set of  $k$  centers in  $\mathbb{R}^d$  can be approximated by the sum of weighted squared distances from the points in the coreset. The output of running an existing clustering algorithm on the coreset would then yield approximation to the output of running the same algorithm on the original data, by the definition of the coreset.

Coresets were first suggested in [2] as a way to improve the theoretical running time of existing algorithms. Moreover, a coreset is a natural tool for handling Big Data using all the computation models that are mentioned in the previous section. This is mainly due to the merge-and-reduce tree approach that was suggested in [19, 7] and is formalized in [14]: coresets can be computed independently for subsets of input points, e.g. on different computers, and then be merged and re-compressed again. Such a binary compression tree can also be computed using one pass over a possibly unbounded streaming set of points, where in every moment only  $O(\log n)$  coresets exist in memory for the  $n$  points streamed so far. Here the coreset is computed only on small chunks of points, so a possibly inefficient coreset construction still yields efficient coreset constructions for large sets; see Fig. 1. Note that the coreset guarantees are preserved while using this technique, while no assumptions are made on the order of the streaming input points.

In practice, this technique can be implemented easily using the map-reduce approach of modern software for Big Data such as Hadoop [1].

The fact that the coreset approximates every set of  $k$  centers, allows us to use the above merge-and-reduce technique where the optimal solution keeps changing, and also to solve the  $k$ -means problem under different constraints or a small set of candidate centers.

## 2 Related Work

We summarize existing coresets constructions for  $k$ -mean queries, as will be formally defined in Section 4.

**Importance Sampling.** Following a decade of research, coreset of size polynomial in  $d$  were suggested in [8]. An improved version of size  $O(dk^2/\varepsilon^2)$  was suggested in [22] which is a special case of the algorithms in [13]. The construction is based on computing an approximation to the  $k$ -mean of the input points (with no constraints on the centers) and then sample points proportionally to their distance to these centers. Each chosen point is then assigned a weight that is inverse proportional to this distance. The probability of failure in this algorithms reduces exponentially with the input size. Coresets of size  $O(dk/\varepsilon^2)$ , i.e., linear in  $k$ , were

suggested in [12], however the weight of a point may be negative or a function of the given query. For the special case  $k = 1$ , such coresets of size  $O(1/\varepsilon^2)$  were suggested in [20] using uniform sampling.

**Projection based coresets.** Coresets of size  $O(k/\varepsilon)$  that are based on projections on low-dimensional subspaces that diminishes the sparsity of the input data were recently suggested in [9] by improving the analysis in [14]. Other type of weak coresets approximates only the optimal solution, and not every  $k$  centers. Such randomized coresets of size independent of  $d$  and only polynomial in  $k$  were suggested in [13] and simplified in [12].

**Deterministic Constructions.** The first coresets for  $k$ -means [19, 18] were based on partitioning the data into cells, and take a representative point from each cell to the coreset, as is done in hashing or Hough transform [5]. However, these coresets are of size at least  $k/\varepsilon^{O(d)}$ , i.e., exponential in  $d$ . Deterministic coreset constructions of size  $k^{O(k/\varepsilon)}$ , i.e., independent of  $d$  but exponential in  $k$ , were suggested in [14] by recursively computing  $k$ -means clustering of the input points. Uniform sampling is then used for replacing each mean of a cluster by a subset of its points. By Markov’s inequality, the probability of failure for these algorithms reduces only linearly with the input size. Therefore they cannot be used in the streaming or distributed setting, where we need to compute union of  $O(n)$  core-sets. Our technique improves this result by suggesting a fully deterministic coreset construction of size  $k^{O(1/\varepsilon^2)}$ , i.e., polynomial in  $k$ .

### 3 Our contribution

Our main technical result is a deterministic algorithm that computes a  $(k, \varepsilon)$ -coreset of size  $k^{O(1)}$  for a set of points  $P$  and every constant error parameter  $\varepsilon > 0$ ; see Theorem 1 and Corollary 1 for details and exact bounds. This is the first coreset which uses number of memory words that is independent of both  $n$  and  $d$ , and only polynomial in  $k$  where each point of  $P$  has  $O(1)$  non-zeroes entries. Using this coreset with the merge-and-reduce technique above, we achieve the following deterministic algorithms:

1. An algorithm that computes a  $(1 + \varepsilon)$  approximation for the  $k$ -means of a set  $P$  that is distributed (partitioned) among  $M$  machines, where each machine needs to send only  $k^{O(1)}$  input points to the main server at the end of its computation.
2. A streaming algorithm that, after one pass over the data and using  $k^{O(1)} \log n$  space returns an  $O(\log n)$ -approximation to the  $k$ -means of  $P$ .
3. Description of how to use our algorithm to boost both the running time and quality of any existing  $k$ -means heuristic using only the heuristic itself, even in the classic off-line setting.
4. Experimental results on real world data-sets and open-code that demonstrate how we applied our coreset to boost the performance of the popular Lloydy’s  $k$ -means heuristic [27, 4].

**Running time.** The overall running time of our algorithm is  $n \log(n) \cdot k^{O(k)}$  for the set  $P$  of  $n$  points seen so far in a possibly unbounded stream, where each point has  $O(1)$  non-zeroes entries. Computing a  $(1 + \varepsilon)$ -approximation to the  $k$ -means on the coreset takes time  $|S|^{O(k)}$  where  $|S|$  is the coreset size. Running time that is exponential in  $k$  is unavoidable for any  $(1 + \varepsilon)$ -approximation algorithm that solves  $k$ -means, even in the planar case ( $d = 2$ ) [26]. Our main contributions thus a coreset construction that uses memory that is independent of  $d$  and running time that is near-linear in  $n$ . This is a new result even for the case  $k = 2$ . To handle large values of  $k$  in our experiments, our algorithm calls popular heuristics instead of computing the optimal  $k$ -means during the coreset construction and on the resulting coreset itself.

### 4 Notation and Main Result

The input to our problem is a set  $P'$  of  $n$  points in  $\mathbb{R}^d$ , where each point  $p \in P'$  includes a multiplicative weight  $u(p) > 0$ . In addition, there is an additive weight  $\rho > 0$  for the set. Formally, a *weighted set* in  $\mathbb{R}^d$  is

a tuple  $P = (P', u, \rho)$ , where  $P' \subseteq \mathbb{R}^d$ ,  $u : P' \rightarrow [0, \infty)$ . In particular, an *unweighted* set has a unit weight  $u(p) = 1$  for each point, and a zero additive weight.

**$k$ -Mean clustering.** For a given set  $Q = \{q_1, \dots, q_k\}$  of  $k \geq 1$  centers (points) in  $\mathbb{R}^d$ , the Euclidean distance from a point  $p \in \mathbb{R}^d$  to its closest center in  $Q$  is denoted by  $\text{dist}(p, Q) = \min_{q \in Q} \|p - q\|_2$ . The sum of these weighted squared distances over the points of  $P$  is denoted by

$$\text{cost}(P, Q) := \sum_{p \in P'} u(p) \cdot \text{dist}^2(p, Q) + \rho,$$

If  $P$  is an unweighted set, this cost is just the sum of squared distances over each point in  $P'$  to its closest center in  $Q$ .

Let  $P'_i$  denote the subset of points in  $P'$  whose closest center in  $Q$  is  $q_i$ , for every  $i \in [m] = \{1, \dots, m\}$ . Ties are broken arbitrarily. This yields a partition  $\{P'_1, \dots, P'_k\}$  of  $P'$  by  $Q$ . More generally, the *partition of  $P$  by  $Q$*  is the set  $\{P_1, \dots, P_k\}$  where  $P_i = (P'_i, u_i, \rho/k)$ , and  $u_i(p) = u(p)$  for every  $p \in P'_i$  and every  $i \in [k]$ .

A set  $Q_k$  that minimizes this weighted sum  $\text{cost}(P, Q)$  over every set  $Q$  of  $k$  centers in  $\mathbb{R}^d$  is called the  *$k$ -mean* of  $P$ . The 1-means  $\mu(P)$  of  $P$  is called the centroid, or the center of mass, since

$$\mu(P) = \frac{1}{\sum_{p' \in P'} u(p')} \sum_{p \in P'} u(p) \cdot p.$$

We denote the cost of the  $k$ -mean of  $P$  by  $\text{opt}(P, k) := \text{cost}(P, Q_k)$ .

**Coreset.** Computing the  $k$ -mean of a weighted set  $P$  is the main motivation of this paper. To this end, we wish to compute another weighed set  $S = (S', w, \phi)$  where  $S'$  is small set (core-set)  $\mathbb{R}^d$  that can be used to approximate  $\text{cost}(P, Q)$  for any set  $Q$  of  $k$  points. In particular, a set  $Q$  that minimizes the weighted cost to  $S$  must also approximately minimize the cost to  $P$ , over such sets  $Q$  in  $\mathbb{R}^d$ .

Formally, let  $\varepsilon > 0$  be an error parameter. The weighted set  $S = (S', w, \phi)$  is a  $(k, \varepsilon)$ -coreset for  $P$ , if for every set  $Q \subset \mathbb{R}^d$  of  $|Q| = k$  centers we have

$$(1 - \varepsilon)\text{cost}(P, Q) \leq \text{cost}(S, Q) \leq (1 + \varepsilon)\text{cost}(P, Q),$$

That is,

$$(1 - \varepsilon) \sum_{p \in P} u(p) \text{dist}^2(p, Q) + \rho \leq \sum_{p \in S} w(p) \cdot \text{dist}^2(p, Q) + \phi \leq (1 + \varepsilon) \sum_{p \in P} u(p) \cdot \text{dist}^2(p, Q) + \rho.$$

To handle streaming data we will need to compute “coresets for union of coresets”, which is the reason that we assume that both the input  $P$  and its coreset  $S$  are weighted sets.

**Sparse coresets.** Unlike previous work, we add the constraints that if each point in  $P$  is sparse, i.e., has few non-zeroes coordinates, then the set  $S$  will also be sparse. Formally, the *maximum sparsity*  $s(P) := \max_{p \in P} \|p\|_0$  of  $P$  is the maximum number of non zeroes entries  $\|p\|_0 = |\{i \mid p_i \neq 0, i \in [d]\}|$  over every point  $p$  in  $P$ .

In particular, if each point in  $S$  is a linear combination of at most  $\alpha$  points in  $P$ , then  $s(S) \leq \alpha \cdot s(P)$ . In addition, we would like that the set  $S$  will be of size independent of both  $n = |P|$  and  $d$ .

We can now state the main result of this paper.

**Theorem 1** (Small sparse coreset). *For every weighted set  $P = (P', u, \rho)$  in  $\mathbb{R}^d$ ,  $\varepsilon > 0$  and an integer  $k \geq 1$ , there is a  $(k, \varepsilon)$ -coreset  $S = (S', w, \phi)$  of size  $|S| = k^{O(1/\varepsilon^2)}$  where each point in  $S'$  is a linear combination of  $O(1/\varepsilon^2)$  points from  $P'$ . In particular, the maximum sparsity of  $S'$  is  $s(P)/\varepsilon^2$ .*

By plugging this result to the traditional merge-and-reduce tree below, it is straight-forward to compute a coreset using one pass over a stream of points. Previous results computed such coresets using  $O(k/\varepsilon)$  points but  $O(dk/\varepsilon)$  memory words (coordinates of points) which is inefficient when, say,  $d \geq n$ . In contrast, the coreset below is computed in memory of size that is independent of  $d$ . Similarly, when the input is distributed among few machines, previous results had to communicate coresets of size at least linear in  $d$  between the machines, while communicating the coreset below will take number of bits that is independent of  $d$ .

**Corollary 1.** *A  $(k, \varepsilon \log n)$  coreset  $(S, w, \phi)$  of size  $|S| = \log(n) \cdot k^{O(1/\varepsilon^2)}$  and maximum sparsity  $s(P)/\varepsilon^2$  can be computed for the set  $P$  of the  $n$  points seen so far in an unbounded stream, using  $|S| \cdot s(P)/\varepsilon^2$  memory words. The insertion time per point in the stream is  $\log(n) \cdot 2^{(k/\varepsilon)^{O(1)}}$ . If the stream is distributed uniformly to  $M$  machines, then the amortized insertion time per point is reduced by a (multiplicative) factor of  $M$  to  $(1/M) \log(n) \cdot 2^{(k/\varepsilon)^{O(1)}}$ . The coreset for the union of streams can then be computed by communicating the  $M$  coresets to a main server.*

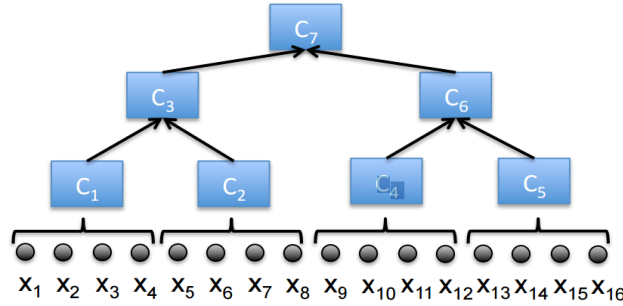


Figure 1: Tree construction for generating coresets in parallel or from data streams [19]. Black arrows indicate “merge-and-reduce” operations. The intermediate coresets  $C_1, \dots, C_7$  are numbered in the order in which they would be generated in the streaming case. In the parallel case,  $C_1, C_2, C_4$  and  $C_5$  would be constructed in parallel, followed by  $C_3$  and  $C_6$ , finally resulting in  $C_7$ . The Figure is from [10].

## 5 Coreset Construction

Our main coreset construction algorithm  $k\text{-MEAN-CORESET}(P, k, \varepsilon)$  gets a set  $P$  as input, and returns a  $(k, \varepsilon)$ -coreset  $(S, w)$ ; see Algorithm 1.

To obtain running time that is linear in the input size, without loss of generality, we assume that  $P$  has  $|P| = k^{O(1/\varepsilon^2)}$  points, and that the cardinality of the output  $S$  is  $|S| \leq |P|/2$ . This is thanks to the traditional merge-and-reduce approach: given a stream of  $n$  points, we apply the coreset construction only on subsets of size  $2 \cdot |S|$  from  $P$  during the streaming and reduce them by half. See Fig. 1 and e.g. [10, 14] for details.

**Algorithm overview.** In Line 1 we compute the smallest integer  $m = k^t$  such that the cost  $\text{opt}(P, m)$  of the  $m$ -means of  $P$  is close to the cost  $\text{opt}(P, mk)$  of the  $(mk)$ -means of  $P$ . In Line 3 we compute the corresponding partition  $\{P_1, \dots, P_m\}$  of  $P$  by its  $m$ -means  $Q_m = \{q_1, \dots, q_m\}$ . In Line 5 a  $(1, \varepsilon)$ -sparse coreset  $S_i$  of size  $O(1/\varepsilon^2)$  is computed for every  $P_i, i \in [m]$ . This can be done deterministically e.g. by taking the mean of  $P_i$  as explained in Lemma 1 or by using a gradient descent algorithm, namely Frank-Wolfe, as explained in [16] which also preserves the sparsity of the coreset as desired by our main theorem. The output of the algorithm is the union of the means of all these coresets, with appropriate weight, which is the size of the coreset.

**Comments.** Line 1 is the only line in the algorithm that takes time exponential in  $k$ . As stated in Section 3 this is unavoidable for a  $(1 + \varepsilon)$ -approximation, where  $\varepsilon$  is an arbitrarily small constant. Still, the memory that is required by our algorithm is polynomial in  $k$ , and the running time is near-linear in  $n$ . In Section 9 this line will be replaced by  $k$ -means heuristic or  $\Omega(1)$ -approximation to yield a practical algorithm for large values of  $k$ .

---

**Algorithm 1:**  $k$ -MEAN-CORESET( $P, k, \varepsilon$ )

---

**Input:** A weighted set  $P$  of points in  $\mathbb{R}^d$ , integer  $k \geq 1$ , and error parameter  $\varepsilon \in (0, 1/4)$ .  
**Output:** A  $(k, \varepsilon)$ -coreset  $S$  for  $P$  that satisfies Theorem 1.

- 1 Compute the smallest integer  $t \geq 0$  such that  $\text{opt}(P, k^t) - \text{opt}(P, k^{t+1}) \leq \varepsilon^2 \cdot \text{opt}(P, k)$   
 /\*  $\text{opt}(P, m)$  is the  $m$ -mean cost of  $P$ . \*/
- 2 Set  $m \leftarrow k^t$
- 3 Set  $\{P_1, \dots, P_m\} \leftarrow$  the partition of  $P$  by  $\text{opt}(P, m)$
- 4 **for**  $i := 1$  **to**  $m$  **do**
- 5     Compute a  $(1, \varepsilon)$ -coreset  $S_i = (S'_i, w_i, \phi_i)$  for  $P_i$
- 6     Set  $w(\mu(S_i)) \leftarrow \sum_{p \in S'_i} w_i(p)$
- 7 Set  $S' \leftarrow \bigcup_{i=1}^m \mu(S_i)$
- 8 Set  $\phi \leftarrow \sum_{i=1}^m \text{cost}(S_i, \mu(S_i))$
- 9 Set  $S \leftarrow (S', w, \phi)$
- 10 **return**  $S$

---

## 6 Proof of Correctness

In this section we prove that a call to  $k$ -MEAN-CORESET( $P, k, \varepsilon$ ) indeed returns a small  $(k, \varepsilon)$ -coreset; see Algorithm 1. We use the variable names as defined in the algorithm (e.g.  $t, m, S$ ) and consider their corresponding values during the last line of the algorithm. We also identify the input weighted set  $P$  by  $P = (P', u, \rho)$ .

The first lemma states the common fact that the sum of squared distances of a set of point to a center is the sum of their squared distances to their center of mass, plus the squared distance to the center (the variance of the set).

**Lemma 1.** For every  $x \in \mathbb{R}^d$

$$\text{cost}(P, x) = \text{cost}(P, \mu(P)) + \|\mu(P) - x\|^2 \sum_{p \in P} u(p).$$

*Proof.* We have

$$\begin{aligned} \text{cost}(P, x) - \rho &= \sum_{p \in P} u(p) \|p - x\|^2 = \sum_{p \in P} u(p) \|(p - \mu(P)) + (\mu(P) - x)\|^2 \\ &= \sum_{p \in P} u(p) \|p - \mu(P)\|^2 + \sum_{p \in P} u(p) \|\mu(P) - x\|^2 + 2(\mu(P) - x) \cdot \sum_{p \in P} u(p)(p - \mu(P)). \end{aligned}$$

The last term equals zero since  $\mu(P) = \frac{1}{\sum_{p' \in P} u(p')} \cdot \sum_{p \in P} u(p) \cdot p$ , and thus

$$\sum_{p \in P} u(p)(p - \mu(P)) = \sum_{p \in P} u(p) \cdot p - \sum_{p \in P} u(p) \mu(P) = \sum_{p \in P} u(p) \cdot p - \sum_{p \in P} u(p) \cdot p = 0.$$

Hence,

$$\text{cost}(P, x) = \rho + \sum_{p \in P} u(p) \|p - \mu(P)\|^2 + \sum_{p \in P} u(p) \|\mu(P) - x\|^2 = \text{cost}(P, \mu(P)) + \|\mu(P) - x\|^2 \sum_{p \in P} u(p).$$

□

The second lemma shows that assigning all the points of  $P$  to the closest center in  $Q$  yields a small multiplicative error if the 1-mean and the  $k$ -mean of  $P$  has roughly the same cost. If  $t = 0$ , this means that we can approximate  $\text{cost}(P, Q)$  using only one center in the query; see Line 1 of Algorithm 1. Note that  $(1 + 2\varepsilon)/(1 - 2\varepsilon) \leq 1 + 4\varepsilon$  for  $\varepsilon < 1/4$ .

**Lemma 2.** *For every set  $Q \subseteq \mathbb{R}^d$  of  $|Q| = k$  centers we have*

$$\text{cost}(P, Q) \leq \min_{q \in Q} \text{cost}(P, \{q\}) \leq \text{cost}(P, Q) \cdot \frac{1 + 2\varepsilon}{1 - 2\varepsilon} + \frac{\text{opt}(P, 1) - \text{opt}(P, k)}{(1 - 2\varepsilon)\varepsilon}. \quad (1)$$

*Proof.* Let  $q^*$  denote a center that minimizes  $\text{cost}(P, \{q\})$  over  $q \in Q$ . The left inequality of (1) is then straight-forward since

$$\begin{aligned} \text{cost}(P, Q) - \min_{q \in Q} \text{cost}(P, \{q\}) &= \sum_{p \in P'} \min_{q \in Q} u(p) \|p - q\|^2 - \sum_{p \in P'} u(p) \|p - q^*\|^2 \\ &= \sum_{p \in P'} u(p) \left( \min_{q \in Q} \|p - q\|^2 - \|p - q^*\|^2 \right) \leq 0. \end{aligned} \quad (2)$$

It is left to prove the right inequality of (1). Indeed, for every  $p \in P'$ , let  $q_p \in Q$  denote the closest point to  $p$  in  $Q$ . Ties are broken arbitrarily. Hence,

$$\min_{q \in Q} \text{cost}(P, \{q\}) - \text{cost}(P, Q) = \sum_{p \in P'} u(p) \|p - q^*\|^2 - \sum_{p \in P'} u(p) \|p - q_p\|^2.$$

Let  $\{P_1, \dots, P_k\}$  denote the partition of  $P$  by  $Q = \{q^1, \dots, q^k\}$ , where  $P_i$  are the closest points to  $q^i$  for every  $i \in [k]$ ; see Section 4. For every  $p \in P'_i$ , let  $q_p^* = \mu(P_i)$ . Hence,

$$\sum_{p \in P'_i} u(p) \|p - q^*\|^2 - \sum_{p \in P'_i} u(p) \|p - q_p\|^2 = \sum_{p \in P'_i} u(p) \|p - \mu(P_i)\|^2 + \|\mu(P_i) - q^*\|^2 \sum_{p \in P'_i} u(p) \quad (3)$$

$$- \left( \sum_{p \in P'_i} u(p) \|p - \mu(P_i)\|^2 + \|\mu(P_i) - q^i\|^2 \sum_{p \in P'_i} u(p) \right) \quad (4)$$

$$= \sum_{p \in P'_i} u(p) \left( \|q_p^* - q^*\|^2 - \|q_p^* - q_p\|^2 \right), \quad (5)$$

where in (3) and (4) we substituted  $x = \mu(P_i)$  and  $x = q_i$  respectively in Lemma 1, and in (5) we use the fact that  $q_p^* = \mu(P_i)$  and  $q_p = q^i$  for every  $p \in P_i$ . Summing (5) over  $i \in [k]$  yields

$$\begin{aligned} \sum_{p \in P'} u(p) \|p - q^*\|^2 - \sum_{p \in P'} u(p) \|p - q_p\|^2 &= \sum_{p \in P'} u(p) \left( \|q_p^* - q^*\|^2 - \|q_p^* - q_p\|^2 \right) \\ &= \sum_{p \in P'} u(p) \left( \|(q_p^* - \mu(P)) + (\mu(P) - q^*)\|^2 - \|(q_p^* - \mu(P)) + (\mu(P) - q_p)\|^2 \right) \end{aligned} \quad (6)$$

$$= \sum_{p \in P'} u(p) \left( \|\mu(P) - q^*\|^2 - \|\mu(P) - q_p\|^2 \right) \quad (7)$$

$$- 2 \sum_{p \in P'} u(p) (q_p^* - \mu(P))(q^* - q_p). \quad (8)$$

To bound (7), we substitute  $x = q^*$  and then  $x = q$  in Lemma 1, and obtain that for every  $q \in Q$

$$\begin{aligned} \left( \|\mu(P) - q^*\|^2 - \|\mu(P) - q\|^2 \right) \sum_{p \in P'} u(p) &= \text{cost}(P, \{q^*\}) - \text{cost}(P, \mu(P)) - (\text{cost}(P, \{q\}) - \text{cost}(P, \mu(P))) \\ &= \text{cost}(P, \{q^*\}) - \text{cost}(P, \{q\}) \leq 0. \end{aligned}$$

where the last inequality is by the definition of  $q^*$ . This implies that for every  $p \in P'$ ,

$$\|\mu(P) - q^*\|^2 - \|\mu(P) - q_p\|^2 \leq 0.$$

Plugging the last inequality in (7) yields

$$\sum_{p \in P'} u(p) \|p - q^*\|^2 - \sum_{p \in P'} u(p) \|p - q_p\|^2 \leq -2 \sum_{p \in P'} u(p) (q_p^* - \mu(P)) (q^* - q_p) \quad (9)$$

$$\leq 2 \sum_{p \in P'} u(p) \|q_p^* - \mu(P)\| \|q^* - q_p\| = \sum_{p \in P'} u(p) \cdot 2 \cdot \frac{\|q_p^* - \mu(P)\|}{\sqrt{\varepsilon}} \cdot \sqrt{\varepsilon} \|q^* - q_p\| \quad (10)$$

$$\leq \sum_{p \in P'} u(p) \left( \frac{\|q_p^* - \mu(P)\|^2}{\varepsilon} + \varepsilon \|q^* - q_p\|^2 \right) \quad (11)$$

$$= \frac{1}{\varepsilon} \sum_{p \in P'} u(p) \|q_p^* - \mu(P)\|^2 + \varepsilon \sum_{p \in P'} u(p) \|q^* - q_p\|^2, \quad (12)$$

where (10) is by Cauchy-Schwartz inequality, and in (11) we use the fact that  $2ab \leq a^2 + b^2$  for every  $a, b \geq 0$ .

To bound the left term of (12) we use the fact  $q_p^* = \mu(P_i)$  and substitute  $x = \mu(P)$ ,  $P = P_i$  in Lemma 1 for every  $i \in [k]$  as follows.

$$\begin{aligned} \sum_{p \in P'} u(p) \|q_p^* - \mu(P)\|^2 &= \sum_{i=1}^k \|\mu(P_i) - \mu(P)\|^2 \sum_{p \in P'_i} u(p) = \sum_{i=1}^k \left( \sum_{p \in P'_i} u(p) \|p - \mu(P)\|^2 - \sum_{p \in P'_i} u(p) \|p - \mu(P_i)\|^2 \right) \\ &= \sum_{p \in P'} u(p) \|p - \mu(P)\|^2 - \sum_{i=1}^k \sum_{p \in P'_i} u(p) \|p - \mu(P'_i)\|^2 \leq \text{opt}(P, 1) - \text{opt}(P, k). \end{aligned} \quad (13)$$

To bound the right term of (12) we use  $(a - b)^2 \leq a^2 + b^2 + 2|ab| \leq 2a^2 + 2b^2$  to obtain

$$\begin{aligned} \sum_{p \in P'} u(p) \|q^* - q_p\|^2 &= \sum_{p \in P'} u(p) \|(q^* - p) + (p - q_p)\|^2 \\ &\leq \sum_{p \in P'} u(p) \left( 2\|q^* - p\|^2 + 2\|p - q_p\|^2 \right) = 2 \cdot (\text{cost}(P, \{q^*\}) + \text{cost}(P, Q)). \end{aligned}$$

Plugging (13) and the last inequality in (12) yields

$$\begin{aligned} \text{cost}(P, \{q^*\}) - \text{cost}(P, Q) &= \sum_{p \in P'} u(p) \|p - q^*\|^2 - \sum_{p \in P'} u(p) \|p - q_p\|^2 \\ &\leq \frac{\text{opt}(P, 1) - \text{opt}(P, k)}{\varepsilon} + 2 \cdot \varepsilon (\text{cost}(P, \{q^*\}) + \text{cost}(P, Q)). \end{aligned}$$

Rearranging,

$$\text{cost}(P, \{q^*\}) \leq \text{cost}(P, Q) \cdot \frac{1 + 2\varepsilon}{1 - 2\varepsilon} + \frac{\text{opt}(P, 1) - \text{opt}(P, k)}{(1 - 2\varepsilon)\varepsilon}$$

Together with (2) this proves Lemma 2.  $\square$



**Lemma 3.** Let  $S$  be a  $(1, \varepsilon)$ -coreset for a weighted set  $P$  in  $\mathbb{R}^d$ . Let  $Q \subseteq \mathbb{R}^d$  be a finite set. Then

$$(1 - \varepsilon) \min_{q \in Q} \text{cost}(P, \{q\}) \leq \min_{q \in Q} \text{cost}(S, \{q\}) \leq (1 + \varepsilon) \min_{q \in Q} \text{cost}(P, \{q\}) \quad (14)$$

*Proof.* Let  $q_P \in Q$  be a center such that  $\text{cost}(P, \{q_P\}) = \min_{q \in Q} \text{cost}(P, \{q\})$ , and let  $q_S \in Q$  be a center such that  $\text{cost}(S, \{q_S\}) = \min_{q \in Q} \text{cost}(S, \{q\})$ . The right side of (14) is bounded by

$$\min_{q \in Q} \text{cost}(S, \{q\}) = \text{cost}(S, \{q_S\}) \leq \text{cost}(S, \{q_P\}) \leq (1 + \varepsilon) \text{cost}(P, \{q_P\}) = (1 + \varepsilon) \min_{q \in Q} \text{cost}(P, \{q\}),$$

where the first inequality is by the optimality of  $q_S$ , and the second inequality is since  $S$  is a coreset for  $P$ . Similarly, the left hand side of (14) is bounded by

$$\begin{aligned} (1 - \varepsilon) \min_{q \in Q} \text{cost}(P, \{q\}) &= (1 - \varepsilon) \text{cost}(P, \{q_P\}) \leq (1 - \varepsilon) \text{cost}(P, \{q_S\}) \leq (1 - \varepsilon)(1 + \varepsilon) \text{cost}(S, \{q_S\}) \\ &= (1 - \varepsilon^2) \min_{q \in Q} \text{cost}(S, \{q\}) \leq \min_{q \in Q} \text{cost}(S, \{q\}). \end{aligned}$$

where the last inequality follows from the assumption  $\varepsilon < 1$ .  $\square$

**Lemma 4.** Let  $S$  be the output of a call to  $k$ -MEAN-CORESET( $P, k, \varepsilon$ ). Then  $S$  is a  $(k, 15\varepsilon)$ -coreset for  $P$ .

*Proof.* By replacing  $P$  with  $P_i$  in Lemma 1 for each  $i \in [m]$  it follows that

$$\text{cost}(P_i, Q) \leq \min_{q \in Q} \text{cost}(P_i, Q) \leq \text{cost}(P_i, Q) \cdot \frac{1 + 2\varepsilon}{1 - 2\varepsilon} + \frac{\text{opt}(P_i, 1) - \text{opt}(P_i, k)}{(1 - 2\varepsilon)\varepsilon}.$$

Summing the last inequality over each  $P_i$  yields

$$\text{cost}(P, Q) \leq \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, Q) \leq \text{cost}(P, Q) \cdot \frac{1 + 2\varepsilon}{1 - 2\varepsilon} + \frac{1}{(1 - 2\varepsilon)\varepsilon} \sum_{i=1}^m (\text{opt}(P_i, 1) - \text{opt}(P_i, k)). \quad (15)$$

Since  $\{P_1, \dots, P_m\}$  is the partition of the  $m$ -means of  $P$  we have  $\sum_{i=1}^m \text{opt}(P_i, 1) = \text{opt}(P, m)$ . By letting  $Q_i$  be the  $m$ -means of  $P_i$  we have

$$\sum_{i=1}^m \text{opt}(P_i, k) = \sum_{i=1}^m \text{cost}(P_i, Q_i) \geq \sum_{i=1}^m \text{cost}(P_i, \cup_{j=1}^m Q_j) = \text{cost}(P, \cup_{j=1}^m Q_j) \geq \text{opt}(P, mk).$$

Hence,

$$\sum_{i=1}^m (\text{opt}(P_i, 1) - \text{opt}(P_i, k)) \leq \text{opt}(P, m) - \text{opt}(P, mk) \leq \varepsilon^2 \text{opt}(P, k) \leq \varepsilon^2 \text{cost}(P, Q),$$

where the second inequality is by Line 1 of the algorithm. Plugging the last inequality in (15) yields

$$\text{cost}(P, Q) \leq \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, Q) \leq \text{cost}(P, Q) \cdot \frac{1 + 3\varepsilon}{1 - 2\varepsilon}. \quad (16)$$

Using Lemma 3, for every  $i \in [m]$

$$(1 - \varepsilon) \min_{q \in Q} \text{cost}(P_i, \{q\}) \leq \min_{q \in Q} \text{cost}(S_i, \{q\}) \leq (1 + \varepsilon) \min_{q \in Q} \text{cost}(P_i, \{q\})$$

By summing over  $i \in [m]$  we obtain

$$(1 - \varepsilon) \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, \{q\}) \leq \sum_{i=1}^m \min_{q \in Q} \text{cost}(S_i, \{q\}) \leq (1 + \varepsilon) \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, \{q\}).$$

By this and Lemma 1

$$(1 - \varepsilon) \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, \{q\}) \leq \text{cost}(S, Q) \leq (1 + \varepsilon) \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, \{q\}).$$

Plugging the last inequality in (16) yields

$$\begin{aligned} (1 - \varepsilon) \text{cost}(P, Q) &\leq (1 - \varepsilon) \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, Q) \\ &\leq \text{cost}(S, Q) \\ &\leq (1 + \varepsilon) \sum_{i=1}^m \min_{q \in Q} \text{cost}(P_i, \{q\}) \leq (1 + \varepsilon) \text{cost}(P, Q) \cdot \frac{1 + 3\varepsilon}{1 - 2\varepsilon} \leq (1 + 15\varepsilon) \text{cost}(P, Q). \end{aligned} \tag{17}$$

Hence,  $S$  is a  $(k, 15\varepsilon)$  coreset for  $P$ .  $\square$

**Lemma 5.** *There is an integer  $t < 1 + 1/\varepsilon^2$  such that*

$$\text{opt}(P, k^t) - \text{opt}(P, k^{t+1}) \leq \varepsilon^2 \cdot \text{opt}(P, k). \tag{18}$$

*Proof.* Contradictively assume that (18) does not hold for every integer  $i < 1 + 1/\varepsilon^2$ . Hence,

$$\text{opt}(P, k) - \text{opt}(P, k^{\lceil 1/\varepsilon^2 \rceil + 1}) = \sum_{i=1}^{\lceil 1/\varepsilon^2 \rceil} (\text{opt}(P, k^i) - \text{opt}(P, k^{i+1})) > \lceil 1/\varepsilon^2 \rceil \cdot \varepsilon^2 \text{opt}(P, k) \geq \text{opt}(P, k).$$

Contradiction, since  $\text{opt}(P, k^{\lceil 1/\varepsilon^2 \rceil + 1}) \geq 0$ .  $\square$

Using the mean of  $P_i$  in Line 5 of the algorithm yields a  $(1, \varepsilon)$ -coreset  $S_i$  as shown in Lemma 1. The resulting coreset is not sparse, but gives the following result.

**Theorem 2.** *There is  $m \leq k^{1/\varepsilon^2}$  such that the  $m$ -means of  $P$  is a  $(k, 15\varepsilon)$ -coreset for  $P$ .*

**Proof of Theorem 1:** We compute  $S_i$  a  $(1, \varepsilon)$  mean coreset for 1-mean of  $P_i$  at line 5 of Algorithm 1 by using Frank-Wolfe [16] algorithm. It follows that  $|S_i| = O(1/\varepsilon^2)$  for each  $i$ , therefore the overall sparsity of  $S$  is  $s(P)/\varepsilon^2$ . This and Lemma 4 concludes the proof.  $\square$

## 7 Why does it work?

In this section we try to give an intuition of why our coreset construction yields a smaller error for the same number of samples, compared to existing coreset constructions. Roughly, this is mainly due to the “cost of independent sampling” that is used by the existing smallest coreset constructions, namely the sensitivity/importance sampling approach [8, 22, 12, 13].

In Fig. 2 the input is a set of 16 points on the plane that is distributed over 7 clusters that are relatively far from each other. Each cluster consists of a single point, except for one cluster that has  $16 - 6 = 10$  points. Given a “budget” (coreset size) of  $m \geq 10$  points, the “optimal coreset” seems to have all the 6 isolated input points, including  $m - 6$  input points inside the large cluster, that are well distributed in this cluster. What would be the expected coresets of size  $m$  using the existing techniques?

**Algorithm  $k$ -Mean-Coreset.** would return exactly this “optimal coreset”, as the  $m$ -means of  $P$  consists of the 6 isolated clusters and the  $m - 6$  means of the large cluster. For the case  $m = 7$ , the algorithm will pick exactly one representative in each cluster, as it is the 7-means of  $P$ . See Fig. 2(left).

**Uniform Sampling.** If the large cluster is sufficiently large, all the points in a uniform sample will be from this cluster, while all the other (singleton) 6 clusters will be missed. Since these clusters are far away

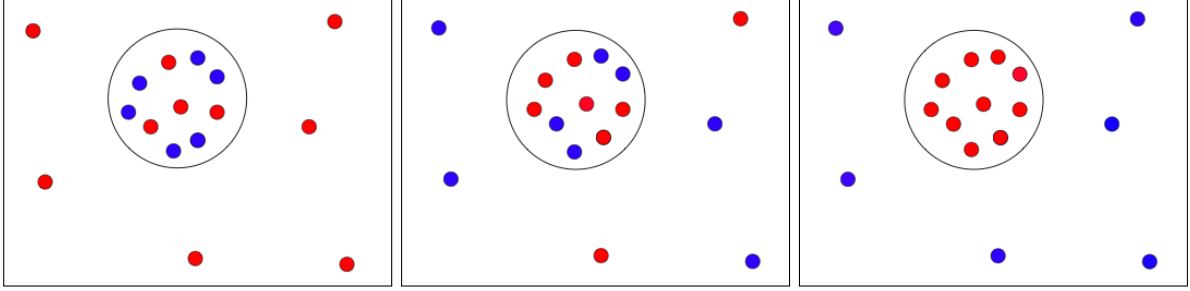


Figure 2: A set of 16 points that consists of 7 clusters that are far from each other. Each of the first 6 clusters contains a single point. The red points are the expected selected points for a coreset of 10 points (with repetitions) using: **(left)** Algorithm 1, **(middle)** Non-uniform (importance/sensitivity) sampling, and **(right)** Uniform sampling.

from each other, the approximation error will then be very large. Even for large sample size, uniform sample misses isolated clusters that are crucial for obtaining a small error. See Fig. 2(right).

**Non-Uniform Sampling.** The optimal distribution that will make sure that a representative from each cluster will be selected to the coreset, is to sample a point from each of the  $k = 7$  clusters with roughly the same probability. However, due to the independent (i.i.d.) sampling approach, the number of samples that are needed in order to have a representative from each cluster is more than  $k = 7$ . In general, the expected sample size is  $O(k \log k)$ . This phenomena is known as the coupon collector problem: if there is a coupon in each box at the supermarket, picked uniformly at random from a collection of  $k$  distinct coupons, then one need to buy  $O(k \log k)$  boxes in expectation to have the collection of all the  $k$  coupons. This is compared to the deterministic construction of Algorithm 1 that always pick the desired  $k$  representatives.

Even after having a representative from each of the isolated clusters a non-uniform sampling will keep sample a point from one of these clusters with probability  $6/7$ . This means that from the total “budget” of  $m$  points in the coreset, a large fraction will be used to sample the same point again and again. This is also why in Fig. 2(middle) there is less number of red points than the other constructions.

## 8 Practical and Simple Boosting of Existing Heuristics

To get the desired phenomena that is described in Section 7 there is no need to actually compute the  $m$ -means for many values of  $m$  and existing heuristics can be used. For example, any reasonable  $k$ -means heuristics for  $k \geq 7$  would yield a set of  $k$  red points as in Fig. 2(left).

**The chicken and the egg phenomena.** As in Algorithm 1, coresets for solving optimization problems usually need to solve the optimization problem in order to decide which points are important to sample. This problem is solved in theory using rough approximation algorithms or bi-criteria approximations [28, 12] that replaces the optimal solution, or using the merge-and-reduce tree that apply the coreset construction only on small sets. In practice, algorithms that compute provable  $(1 + \epsilon)$ -approximations or even 2-approximations for the  $k$ -means clustering are rarely used. Instead, heuristics such as the classic LLoyd’s  $k$ -means or  $k$ -Means++ [27, 4] are used.

Based on our experimental results, a rough approximation using existing heuristics seems to be suffices. In addition, plugging  $\epsilon$  as an input, almost always produces coreset with error that is much smaller than  $\epsilon$ . This is common also in other coresets and related to the facts that the analysis is (i) for the worst case input set and not a specific  $P$  that is usually well structured, (ii) sub-optimal compared to the actual error, (iii) consider every set of  $k$  centers, while we usually care about the optimal solution under some constraints.

We suggest to take the coreset size  $|S|$  as the input and run  $m = 1, \dots, |S|$  iterations of our algorithm. In fact, our experimental results suggest the following simple approach that use a single instead of  $|S|$  runs and yields only slightly less better results.

**Boosting technique.** Given a heuristic for solving the  $k$ -means problem using some  $N \gg k$  iterations, Algorithm 1 suggests to run the heuristic only small number of  $m = O(k) \ll N$  iterations. Then, we take the mean of each of the  $m$  clusters, weighted by the size of the cluster, or construct a  $(1, \varepsilon)$ -coresets on each of the  $m$  clusters. Then, we run the heuristic  $N$  times on this “coreset” of size  $m$ . Even if each iteration of the heuristics takes linear time of  $O(nd)$ , the running time is reduced from  $O(Nnd)$  to  $O(mnd) = O(knd)$ .

**Example 1: Llyod’s  $k$ -means.** In the case of Lloyd’s  $k$ -means, each iteration takes  $O(nd)$  time for computing the distances from the existing seed of  $k$  centers, and then  $O(nd)$  time is needed to compute the next set of centers. The boosting technique above suggests to run  $m \sim O(k) \ll N$  such iterations on  $P$  to produce a weighted coreset of size  $m$ . Then run the  $N$  iterations on this coreset.

**Example 2: KMean++.** The KMean++ algorithm picks another point to the output set in each iteration, where the first point is a random seed. The next point is sampled with probability that is proportional to the distances of the input points to the center (points) that were already picked. This is very similar to the importance sampling that is used for constructing existing coresets with a crucial difference: the sampling is not independent and same for each new point, but *adaptive*, i.e., based on points that were already picked. This is exactly the advantage of our approach compared to the non-uniform sampling, as described in Fig. 2 and Section 7. Note that KMean++ will always select the right centers in Fig. 2, no matter what is the seed and although it is a random algorithm.

The KMean++ algorithm is very natural for using with our boosting technique: We just run it for  $m$  iterations to get a coreset of size  $m$ . Then we run KMean++  $N \gg m$  times on the coreset. In each of the  $N$ th times we use a different seed (first point) and take the optimal among the  $N$  sets of  $k$ -mean candidates. Line 3 of Algorithm 1 suggests an interesting way to choose the size  $m$  of the coreset, based on our analysis in the supplementary material.

## 9 Experimental Results

**Datasets.** To produce experimental results we have use two well known datasets.

**MNIST handwritten digits[24].** The MNIST dataset consist of  $n = 60,000$  grayscale images of handwritten digits. Each image of size  $28 \times 28$  pixels was transformed to the the vector row of  $d = 784$  dimensions.

**Pendigits[3].** This is a dataset from the UCI repository. The dataset created out of 250 samples provided by 44 writers. These writers were asked to write 250 digits in random order inside boxes of  $500$  by  $500$  tablet pixel resolution. The tablet sends  $x$  and  $y$  tablet coordinates and pressure level values of the pen at fixed time intervals (sampling rate) of 100 miliseconds. Digits are represented as constant length feature vectors of size  $d = 16$  the number of digits in the dataset is  $n = 10992$ .

**NIPS dataset[23].** The OCR data from the collection which represents 13 years of NIPS proceedings. The overall of 15,000 pages and 1958 articles. For each author there is a words count vector extracted, where  $i$ th entry in the vector represents count of the particular word which was used in one of the conference submissions by given author. There are overall  $n = 2865$  authors and words corpus size is  $d = 14036$ .

**Expirement.** We used our algorithm to boost the performance of Lloyd’s  $k$ -means heuristic as explained in Section 8. Give a coreset size  $m$  we run this heuristic for only 3 iterations with  $m$  centers. We compared our algorithm with uniform and importance sampling algorithms using both offline computation setting and streaming data model. For offline computation we used datasets above to produce coresets of sizes  $100 \leq m \leq 1500$ , then computed  $k$ -means with values of  $k = 10, 15, 20, 25$  using Lloyd’s heuristic for many iterations till convergence. While to simulate streaming data model we divided datasets into chunks and computed coresets of sizes  $10 \leq m \leq 500$  using map-and-reduce techniques to construct a coreset tree, later repeated computation of  $k$ -means for same values of  $k$ .

For each set of  $k$  centers that was produced, we computed sum of squared distances to the original (full) set of points, and denoted these “approximated solutions” by  $C_1, C_2$  and  $C_3$  for uniform, non uniform sampling and our algorithm respectively. The “ground truth” or “optimal solution”  $C^k$  was computed using  $k$ -means on entire dataset until convergence. The empirical estimated error  $\varepsilon$  is then defined to be  $\epsilon = C_t/C_k - 1$  for coreset number  $t = 1, 2, 3$ .

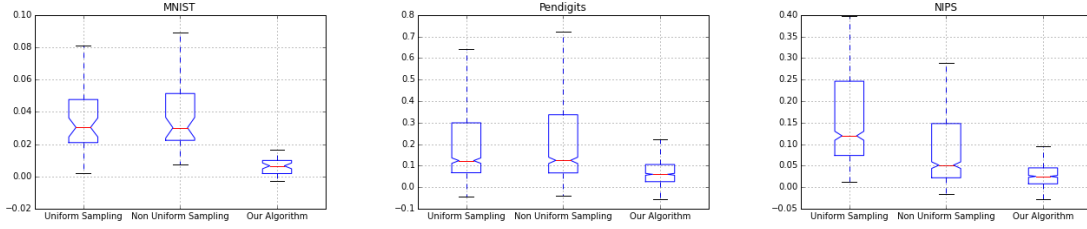


Figure 3: *Error (y-axis) box-plots for real-data sets, offline computation model.*

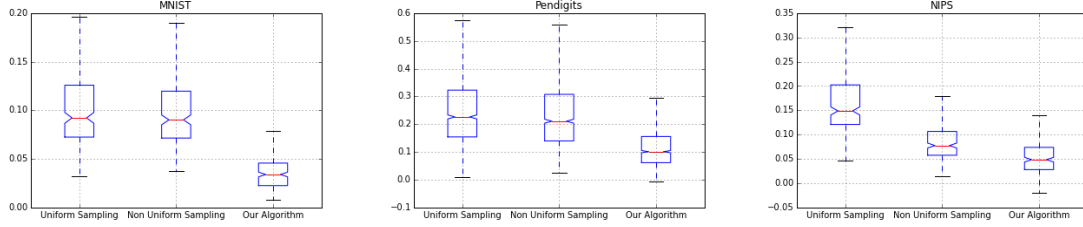
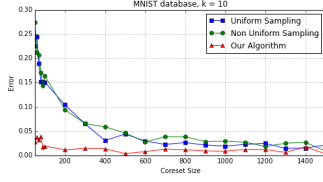


Figure 4: *Error (y-axis) box-plots for real-data sets, streaming computation model.*

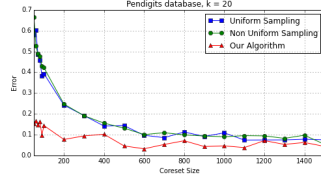
**Results for datasets** Fig.5 and Fig.6 shows results for offline setting and streaming models respectively. The results of our algorithm outperform the uniform sampling and non-uniform sampling algorithms. Important to note, that our algorithm starts with very small error value compared to others and improves error value gradually with sample size, while the other two start with greater error values and succeed to converge to significantly smaller values only for large sample subsets.

Fig.3 and Fig.4, show the boxplot of error distribution for all three algorithms in offline and streaming settings. It's easy to see that our algorithm shows little variance across all experiments and mean error value is very close to the median, indicating that our algorithm produces very stable results, while running on streaming data whiskers are broader due to the multiplicative factor of  $\log n$ .

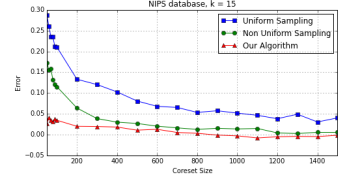
In Fig.7 we present the memory (RAM) footprint during the coreset construction based on synthetically generated random data. These results are common to other coreset papers. The oscillations correspond to the number of coresets in the tree that each new chunk needs to update. For example, the first point in a streaming tree is updated in  $O(1)$ , however the  $2^i$ th point for some  $i \geq 1$  climbs up through  $O(\log i)$  levels in the tree, so  $O(\log i)$  coresets need to be merged.



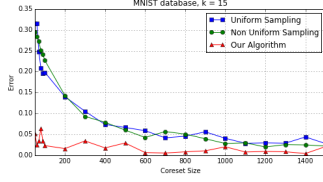
(a) MNIST,  $k=10$



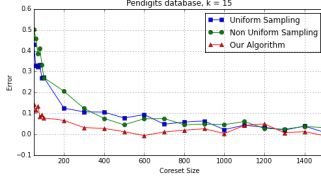
(b) Pendigits,  $k=10$



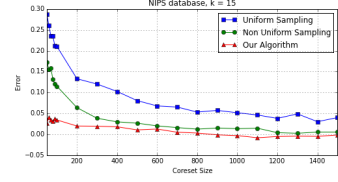
(c) NIPS,  $k=5$



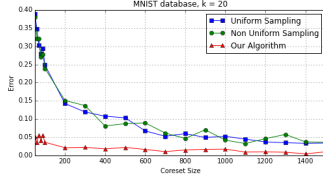
(d) MNIST,  $k=15$



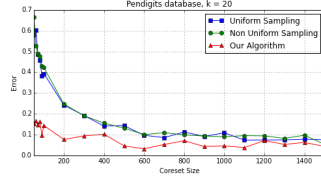
(e) Pendigits,  $k=15$



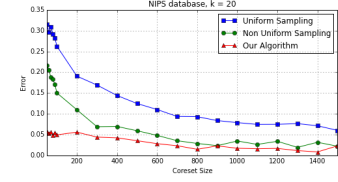
(f) NIPS,  $k=10$



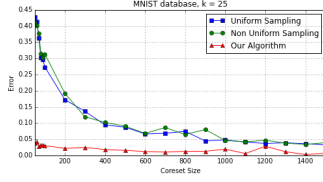
(g) MNIST,  $k=20$



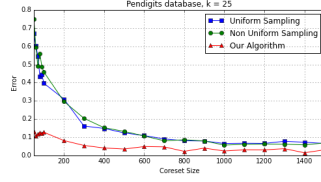
(h) Pendigits,  $k=20$



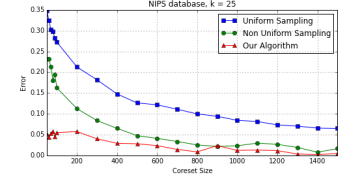
(i) NIPS,  $k=15$



(j) MNIST,  $k=25$



(k) Pendigits,  $k=25$



(l) NIPS,  $k=20$

Figure 5: *Offline setup comparison of uniform sampling, non uniform sampling and our algorithms.*

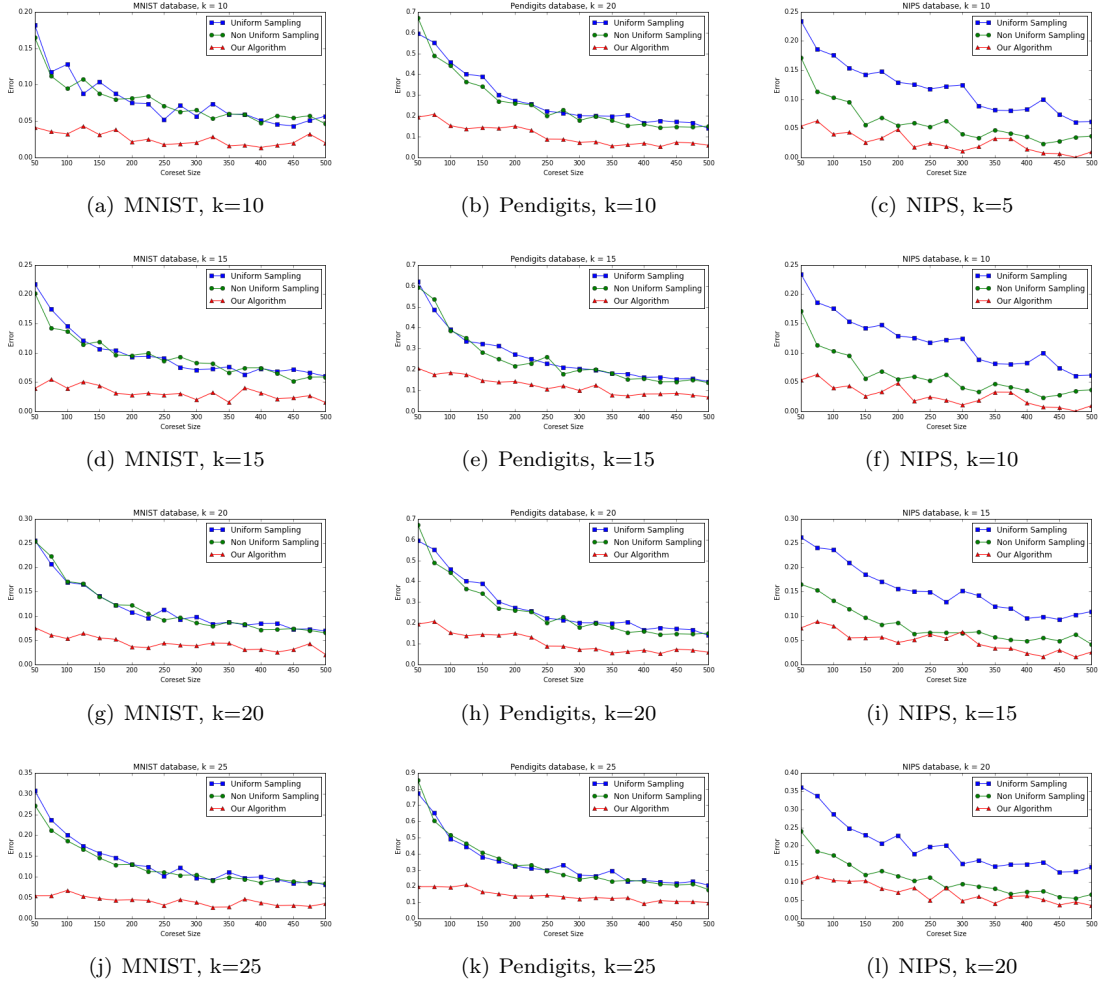


Figure 6: *Streaming setup comparison of uniform sampling, non uniform sampling and our algorithms.*

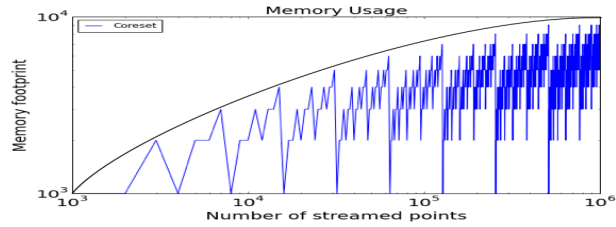


Figure 7: *Allocated memory (y-axis) grows logarithmically during streaming coreset construction. The Zig-zag patterns caused by the binary merge-reduce tree in Fig. 1.*

## References

- [1] Hadoop apache, <http://hadoop.apache.org>, 2009.
- [2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *Journal of the ACM*, 51(4):606–635, 2004.
- [3] F. Alimoglu, D. Doc, E. Alpaydin, and Y. Denizhan. Combining multiple classifiers for pen-based handwritten digit recognition. 1996.
- [4] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.
- [5] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.
- [6] A. Barger and D. Feldman. Kmeans coreset code, <http://sites.hevra.haifa.ac.il/bdrl/software/>, 2015.
- [7] J. L. Bentley and J. B. Saxe. Decomposable searching problems i: Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.
- [8] K. Chen. On  $k$ -median clustering in high dimensions. In *Proc. 17th Annu. ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 1177–1185, 2006.
- [9] M. Cohen, S. Elder, C. Musco, C. Musco, and M. Persu. Dimensionality reduction for  $k$ -means clustering and low rank approximation. In *(STOC)*, 2015.
- [10] D. Feldman, M. Faulkner, and A. Krause. Scalable training of mixture models via coresets. In *Advances in Neural Information Processing Systems*, pages 2142–2150, 2011.
- [11] D. Feldman, A. Fiat, and M. Sharir. Coresets for weighted facilities and their applications. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 315–324. IEEE, 2006.
- [12] D. Feldman and M. Langberg. A unified framework for approximating and clustering data. In *STOC*, pages 569–578, 2011. See <http://arxiv.org/abs/1106.1379> for fuller version.
- [13] D. Feldman, M. Monemizadeh, and C. Sohler. A PTAS for  $k$ -means clustering based on weak coresets. In *SoCG*, 2007.
- [14] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for  $k$ -means, PCA and projective clustering. In *SODA*, 2013.
- [15] D. Feldman, M. Schmidt, and C. Sohler. Turning big data into tiny data: Constant-size coresets for  $k$ -means, pca and projective clustering. In *Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1434–1453. SIAM, 2013.
- [16] D. Feldman, M. V. Volkov, and D. Rus. Dimensionality reduction of massive sparse datasets using coresets. *CoRR*, abs/1503.01663, 2015.
- [17] H. W. Hamacher and Z. Drezner. *Facility location: applications and theory*. Springer Science & Business Media, 2002.
- [18] S. Har-Peled and A. Kushal. Smaller coresets for  $k$ -median and  $k$ -means clustering. *Discrete Comput. Geom.*, 37(1):3–19, 2007.
- [19] S. Har-Peled and S. Mazumdar. On coresets for  $k$ -means and  $k$ -median clustering. In *STOC*, 2004.



- [20] Inaba, Katoh, and Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k-clustering. In *SoCG*, pages 332–339, 1994.
- [21] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.
- [22] M. Langberg and L. J. Schulman. Universal  $\epsilon$  approximators for integrals. *SODA*, 2010.
- [23] Y. LeCun. Nips online web site. <http://nips.djvuzone.org>, 2001.
- [24] Y. LeCun and C. Cortes. The mnist database of handwritten digits, 1998.
- [25] J. A. Lee and M. Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.
- [26] M. Mahajan, P. Nimbhorkar, and K. Varadarajan. The planar k-means problem is np-hard. In *WALCOM*, pages 274–285. Springer, 2009.
- [27] R. Ostrovsky, Y. Rabani, L. J. Schulman, and C. Swamy. The effectiveness of lloyd-type methods for the k-means problem. In *FOCS'06*, pages 165–176. IEEE, 2006.
- [28] K. Varadarajan and X. Xiao. Greedy minimization of weakly supermodular set functions. <http://arxiv.org/abs/1502.06528>, 2015.